Rank Flow:
Fluid Simulation through MPI

Aiden Ferguson

Problem Description/Background

Classical Mechanics

To begin, we consider Newton's (paraphrased) 2nd law:

The net Force on a body is the body's acceleration multiplied by mass. Equivalently, this is the rate at which momentum changes over time.

What this means, practically, is that any calculation for forces on a body must take into account the instantaneous mass and acceleration of that body.

This gives a path for derivation with fluids.

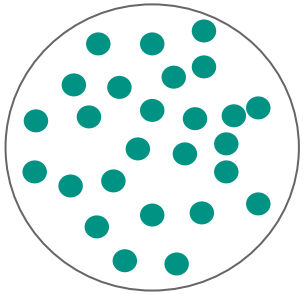
$$F = ma$$

Classical Mechanics

We have immediately run into a problem here, though: fluids aren't singular bodies!

What we do instead is consider the fluid as a set of small/no mass particles. This allows us to consider actions on these singular bodies, and connect them to each other to develop behavior for the whole fluid.

Solid



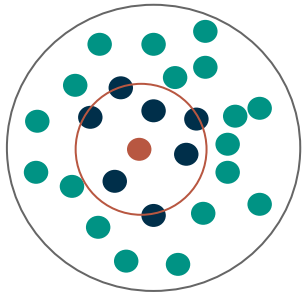
Simulated Fluid

$$F = ma$$

Warning, Non-exact Mathematical Equivalence Follows

Substituting Mass

Using this particle view of fluid then, we can account for the term of mass of a fluid using **density**, utilizing the factor of Volume (or Area) then when calculating Force and Acceleration.



Possible Density Calculation

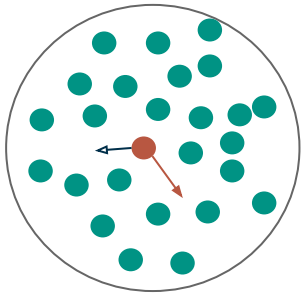
$$F = ma$$

$$m : \rho = mV$$

Substituting Acceleration

Acceleration, then, is accounted for relatively simply by velocity and its change over time, especially for a given singular particle.

We will say it is accounted for by the change in velocity over time (acceleration itself), and the instantaneous actual velocity of the fluid/particle.



$$F = ma$$

$$m : \rho = mV$$

$$a : \frac{\partial u}{\partial t} + (u \cdot \nabla)u$$

Substituting Force (Sum of Forces)

Force, or the sum of the forces that exist within the fluid, is the portions relating **Body**, **Viscosity**, and **Pressure**.

Body refers to primarily Gravity, the self explanatory acceleration towards the Earth.

Viscosity is the force exerted by the fluid's "stickiness" (for lack of a better term) to itself. With a particle model, this is when those particles collide/rub against each other.

Pressure is a repellant force, loosely the inverse towards Viscosity. With Particles, pressure is how much they push each other apart.

$$F = ma$$

$$m : \rho = mV$$

$$a : \frac{\partial u}{\partial t} + (u \cdot \nabla)u$$

$$F : -\nabla p + \nu \nabla^2 u + f$$

The Result: the Navier-Stokes Equation

Substituting the portions in to the Newtonian Force equation (plus mathematical adjustments), we have the resulting Navier-Stokes Equation.

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = \nu\nabla^2 u - \frac{1}{\rho}\nabla p + \frac{1}{\rho}f$$

Alternative Form:

$$\rho\left(\frac{\partial u}{\partial t} + (u \cdot \nabla)u\right) = \mu\nabla^2 u - \nabla p + f$$

Body Forces (Gravity)

Density: $\rho = mV$

Velocity: u

Dynamic Viscosity: μ

Kinematic Viscosity: $\nu = \frac{\mu}{\rho}$

Gradient: ∇

Applying to Code, and Simulation

Simulation Model

As mentioned earlier, the thought process around fluid is considering it as a collection of smaller particles as opposed to one uniform structure. This allows for more dynamic behavior, and gives an easier model for computation.

The Navier-Stokes Equation, then, gives a guide for what to compute for these individual fluid particles.

$$\rho\left(\frac{\partial u}{\partial t} + (u \cdot \nabla)u\right) = \mu\nabla^2 u - \nabla p + f$$

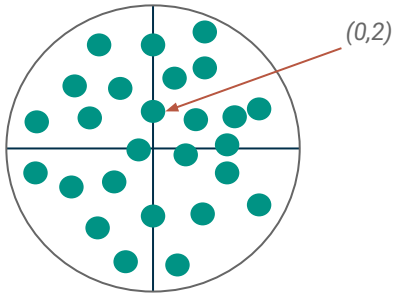
Simulation Model

As a starting point, we must know the position of individual particles in a simulation. This is a given value at the start of a simulation, and must be calculated at each time-step for simulation display.

After all, we have to display the particles for the simulation.

Combined with the aforementioned timestep, we can estimate positional change over time (**velocity**), and velocity change over time (**acceleration**).

$$\rho\left(\frac{\partial u}{\partial t} + (u \cdot \nabla)u\right) = \mu\nabla^2 u - \nabla p + f$$



Simulation Model

Other portions, then, must be calculated at individual timesteps.

The primary calculation of concern is the **density** of the fluid when in the neighborhood of individual particles.

Values for **viscosity** (μ) and **pressure** (p) can be set constants; inherent behavior of the fluid. How they affect a particle at a particular instant, however, is determined by the instantaneous density of the fluid.

$$\rho\left(\frac{\partial u}{\partial t} + (u \cdot \nabla)u\right) = \mu\nabla^2 u - \nabla p + f$$

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = \nu\nabla^2 u - \frac{1}{\rho}\nabla p + \frac{1}{\rho}f$$

Simulation Model

And finally, **body forces** (***f***) are also determined as a factor of density.

While the primary component here is the gravitational force, we must also remember Newton's 3rd law:

Each action has an equal and opposite reaction.

The forces an individual particle exerts on other particles, by way of pressure and viscosity, will have an equally opposing effect on the particle itself.

This must be accounted for.

$$\rho\left(\frac{\partial u}{\partial t} + (u \cdot \nabla)u\right) = \mu\nabla^2 u - \nabla p + f$$

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = \nu\nabla^2 u - \frac{1}{\rho}\nabla p + \frac{1}{\rho}f$$

Computational Model

We now have a bill of needs/features to include and calculate within our computation.

There are **Global Constants**:

- μ: Viscosity
- p: Pressure
- Some gravitational acceleration/force
- ∂t: Time Change

Local values to track per particle:

- Position, u: Velocity

And particle calculations at every step:

- ρ: Density, and Forces based off of it.

$$\rho\left(\frac{\partial u}{\partial t} + (u \cdot \nabla)u\right) = \mu\nabla^2 u - \nabla p + f$$

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = \nu\nabla^2 u - \frac{1}{\rho}\nabla p + \frac{1}{\rho}f$$

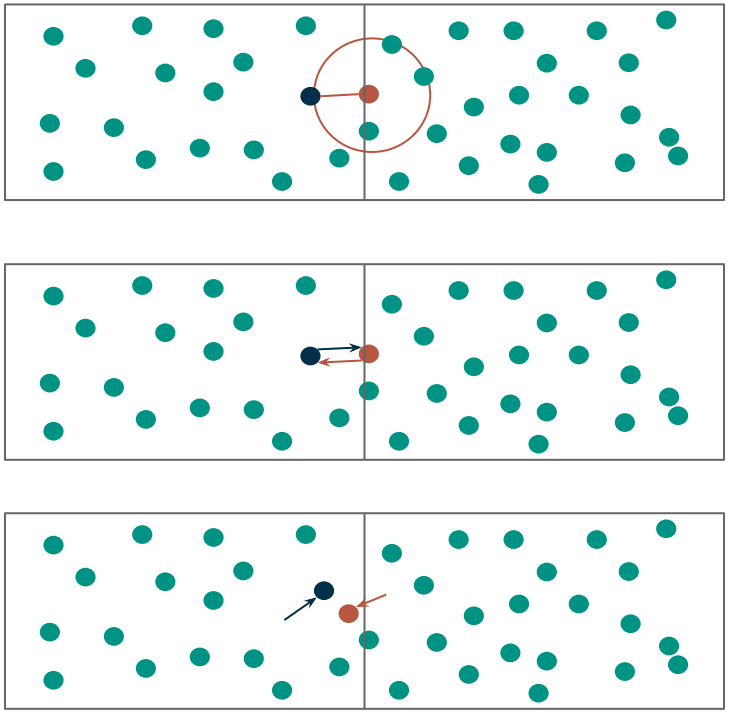
Computational Model - Parallel

Now how do we simulate in a parallel environment?

The approach I have taken is splitting by Volume(or area in 2D), and as result assigning responsibility for particles to the MPI ranks based off their location.

In particular, this means **communication** through MPI occurs in three distinct phases:

1. Communicating positions for density calculation
2. Communicating Reactionary forces off of particles within locality to each other
3. Communicating particle transfer to new Ranks

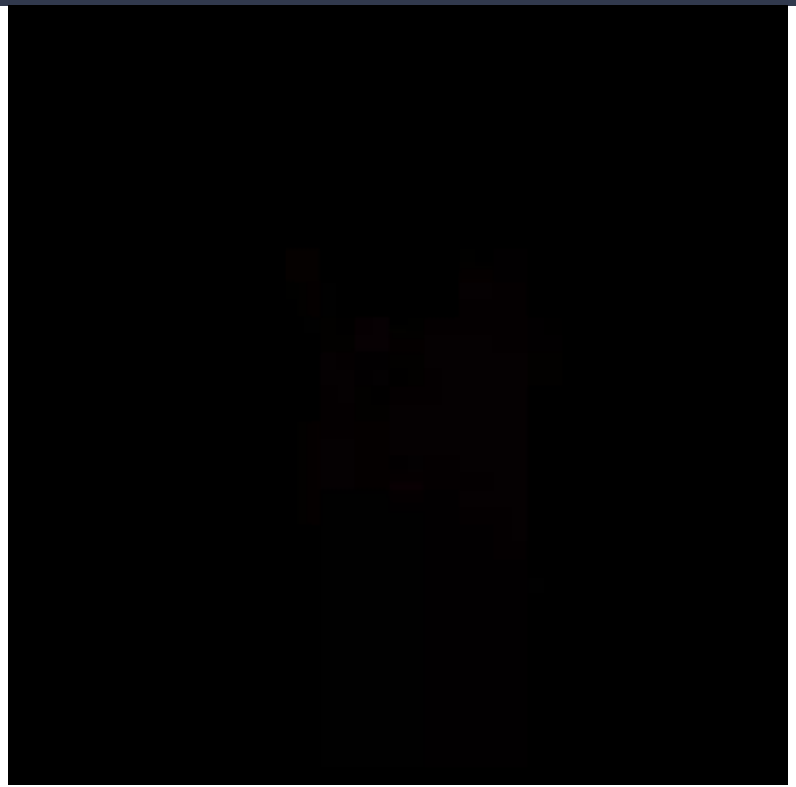


Computational Model - Parallel Pseudocode

```
MPI_Init() // Plus other background needs, eg Comm Size and Rank
...
particles[particleCount]
// Initialize particles for the region the rank oversees
// Particles given initial position
for(timeStep; timeStep < timeMax; timeStep++)
  for(particlesInDomain) // Loop through the particles the Rank controls
    Calculate if density data needed from other ranks

  // We are sending our request for data for particular positions to other ranks
  // which cover those areas of the overall graph/volume
  MPI_SendAll(Positions/Radius where data is needed)
  MPI_RecvAll(Receive requests for data about own particles)
  // Then send requested data back, receive your own requests in term, if there
  // were any
  MPI_SendAll(Requested data from others)
  MPI_RecvAll(Position data requests)

  for(particlesInDomain)
    Calculate Density in radius
    Calculate net forces for the individual particle
    Save Newtonian reaction for particles which influenced the individual particle
    // Note that these forces could be on just local particles for this rank, but we
    // need to ensure that particles in other nodes have needed data too

  MPI_SendAll(Reaction forces for other particles)
  MPI_RecvAll(Reaction forces on your own particles)

  for(particlesInDomain)
    Now calculate final forces, combining initial forces (viscosity and pressure)
    and reactionary forces from other particles

    Through this, calculate new velocity for particle this timeStep

    Calculate new position
    Print/Output the Position
    Calculate if particle has now moved to Area/Volume domain of another Rank

  MPI_SendAll(Particles Transferring to other Rank)
  MPI_RecvAll(Particles Transferring to this Rank)

MPI_Finalize()
```

Results

Animated Results - A Steady Progression

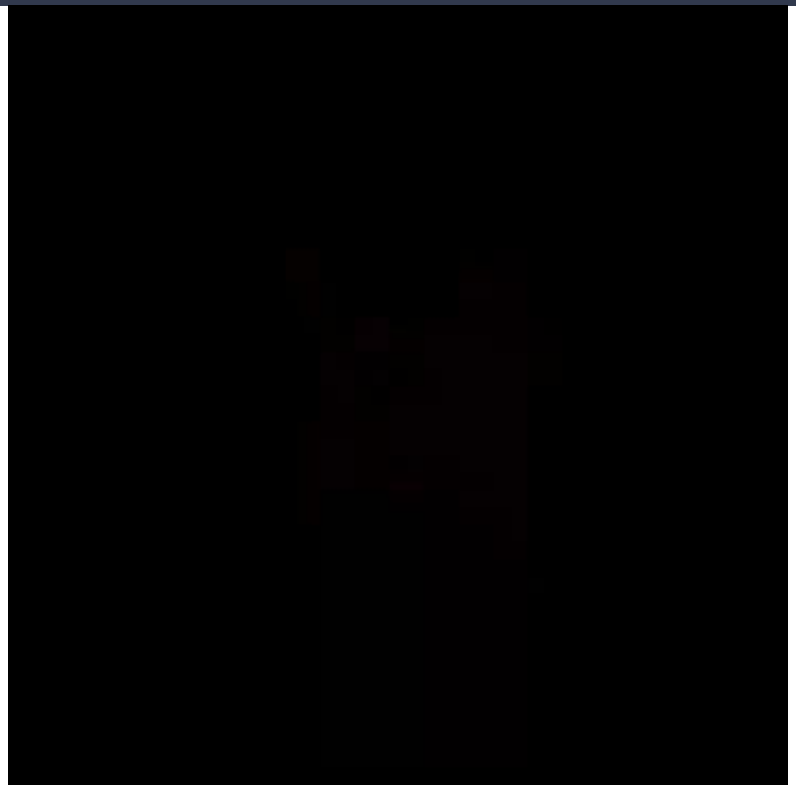


Initial animation results were extremely sparse:

- 1000 x 1000 x 1000 volume
- Only 10,000-20,000 particles

Particles essentially never interacted with each other.

Animated Results - A Steady Progression

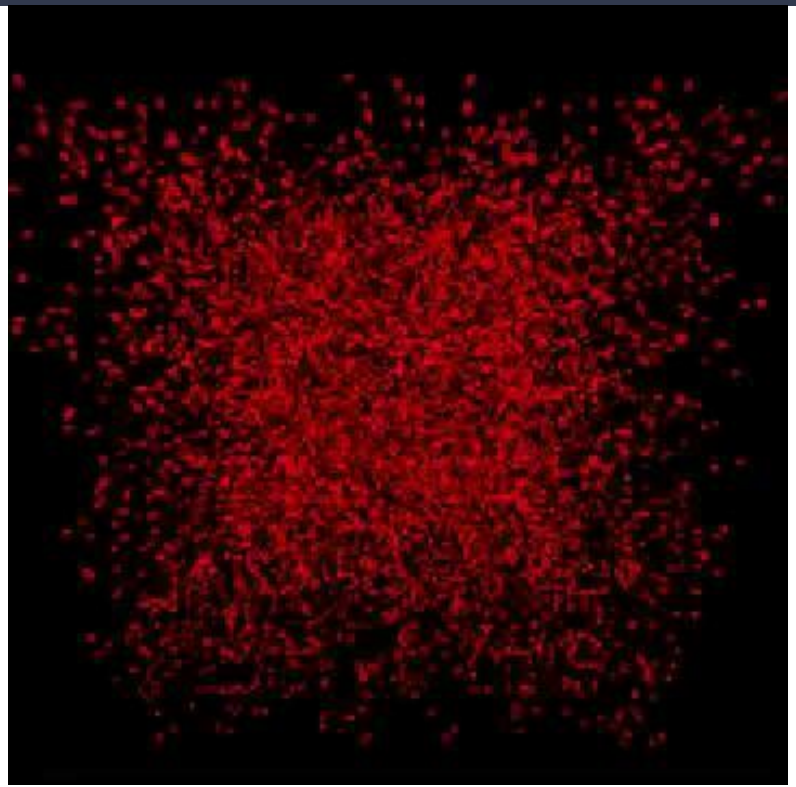


Initial animation results were extremely sparse:

- 1000 x 1000 x 1000 volume
- Only 10,000-20,000 particles

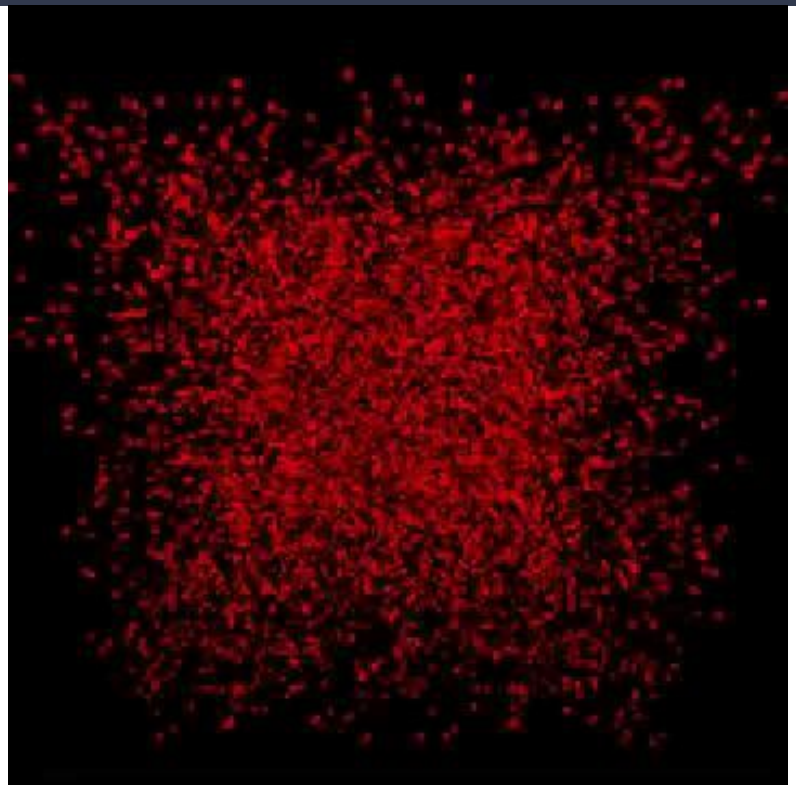
Particles essentially never interacted with each other.

Animated Results - A Steady Progression



Condensing a similar particle count (~10,000) into a much smaller volume (100 x 100 x 100) yielded actually visible results.

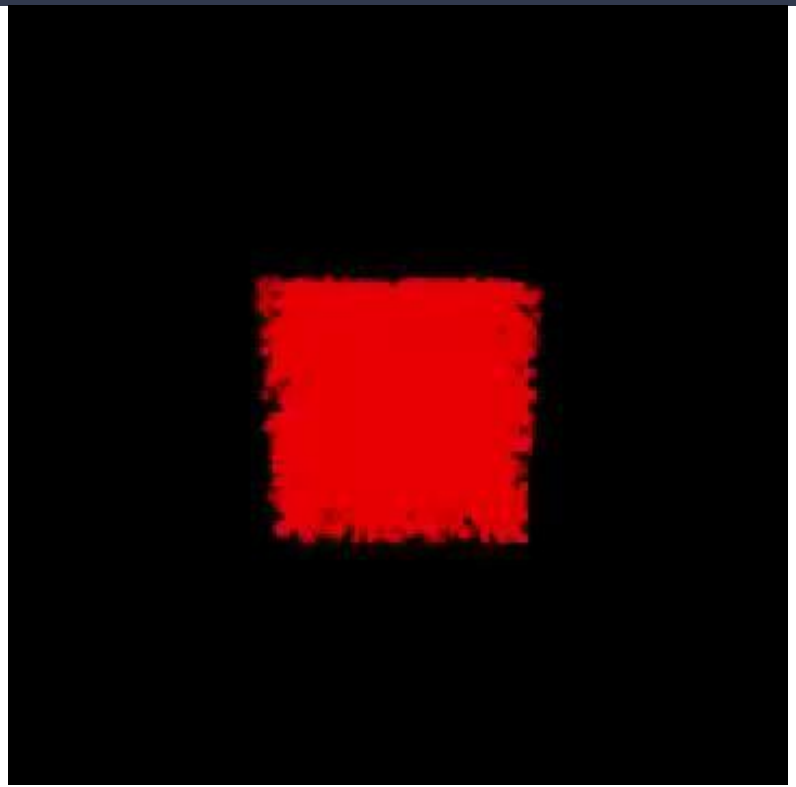
Animated Results - A Steady Progression



Now that particles were visible, modifying simulation variables concerning pressure, gravity, the radius of effect between particles improved the liquidity of results.

Albeit, results still appear rather gaseous.

Animated Results - A Steady Progression



Further condensing the same particle count (~10,000) into an even smaller volume of 50 x 50 x 50 gives even better visuals.

Because the particles were consistently given 1 unit diameters, volumes of this size ensured multiple particle layers in the simulated fluid.

This volume and particle-size ratio was what determined quantities for performance testing.

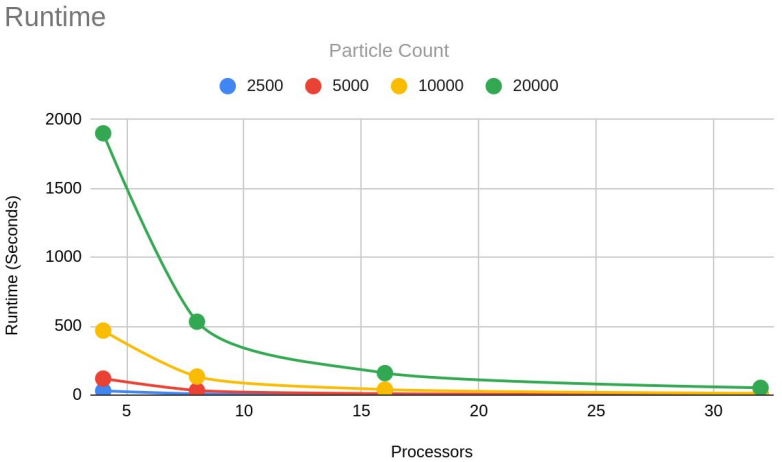
Runtime Results

Results from simulation with:

- 1 Computational Node (Intel Gold 6230, 40 Cores at 2.5 GHz)
- 50 x 50 x 50 Fluid Bounds Box (To force computationally significant proximity)

Results Averaged across 3 individual runs

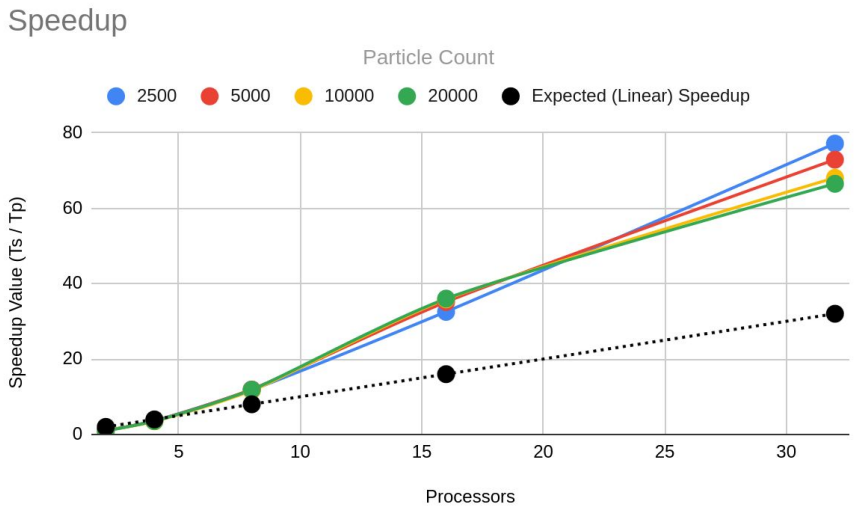
Particles	Processors				
	2	4	8	16	32
2500	29.95503333	8.2524	2.54229	0.920911	0.3884446667
5000	119.7816667	33.56783333	10.03533333	3.41136	1.644106667
10000	468.2033333	134.5213333	40.32113333	13.07033333	6.880236667
20000	1899.666667	532.5916667	159.797	52.73093333	28.57003333



Runtime Analysis

Speedup		Processors			
Parts	2	4	8	16	32
2500	1	3.629857173	11.78269723	32.52760944	77.1153163
5000	1	3.568346681	11.93599283	35.11258462	72.85516755
10000	1	3.480513624	11.61185946	35.82183571	68.05046919
20000	1	3.56683513	11.88799957	36.02565983	66.49158034

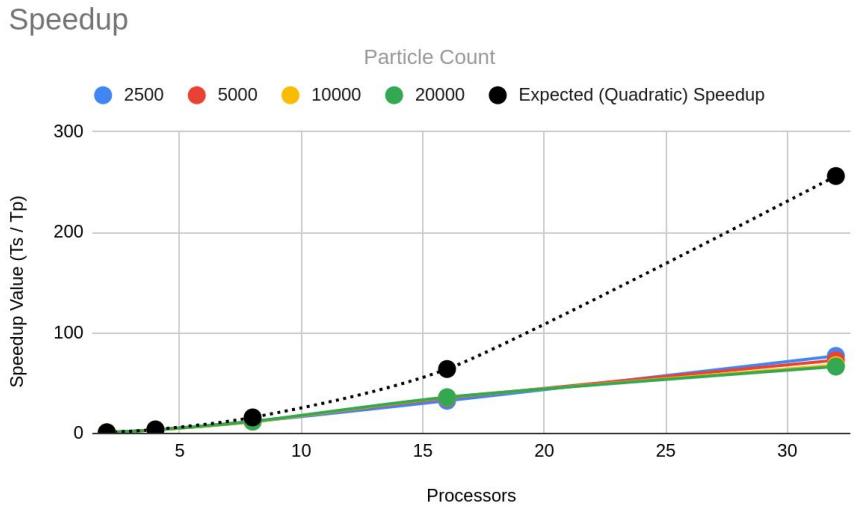
Efficiency		Processors			
Parts	2	4	8	16	32
2500	1	1.814928586	2.945674307	4.06595118	4.819707268
5000	1	1.784173341	2.983998206	4.389073077	4.553447972
10000	1	1.740256812	2.902964864	4.477729464	4.253154325
20000	1	1.783417565	2.971999892	4.503207478	4.155723771



Runtime Analysis

Speedup		Processors			
Parts	2	4	8	16	32
2500	1	3.629857173	11.78269723	32.52760944	77.1153163
5000	1	3.568346681	11.93599283	35.11258462	72.85516755
10000	1	3.480513624	11.61185946	35.82183571	68.05046919
20000	1	3.56683513	11.88799957	36.02565983	66.49158034

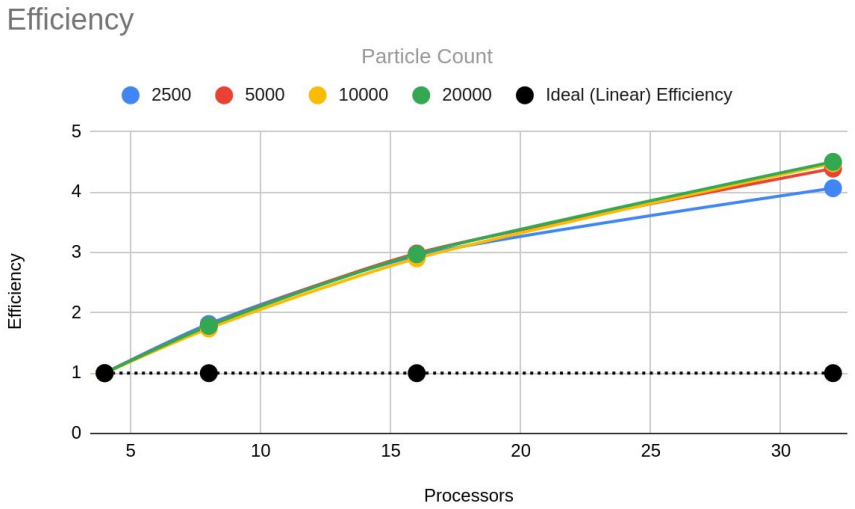
Efficiency		Processors			
Parts	2	4	8	16	32
2500	1	1.814928586	2.945674307	4.06595118	4.819707268
5000	1	1.784173341	2.983998206	4.389073077	4.553447972
10000	1	1.740256812	2.902964864	4.477729464	4.253154325
20000	1	1.783417565	2.971999892	4.503207478	4.155723771



Runtime Analysis

Speedup		Processors			
Parts	2	4	8	16	32
2500	1	3.629857173	11.78269723	32.52760944	77.1153163
5000	1	3.568346681	11.93599283	35.11258462	72.85516755
10000	1	3.480513624	11.61185946	35.82183571	68.05046919
20000	1	3.56683513	11.88799957	36.02565983	66.49158034

Efficiency		Processors			
Parts	2	4	8	16	32
2500	1	1.814928586	2.945674307	4.06595118	4.819707268
5000	1	1.784173341	2.983998206	4.389073077	4.553447972
10000	1	1.740256812	2.902964864	4.477729464	4.253154325
20000	1	1.783417565	2.971999892	4.503207478	4.155723771



Runtime Analysis

Particles	Processors				
	2	4	8	16	32
2500	29.95503333	8.2524	2.54229	0.920911	0.3884446667
5000	119.7816667	33.56783333	10.03533333	3.41136	1.644106667
10000	468.2033333	134.5213333	40.32113333	13.07033333	6.880236667
20000	1899.666667	532.5916667	159.797	52.73093333	28.57003333

	Processors				
Particles	2	4	8	16	32
2500		8.33779	2.601093333	1.008166667	0.4703343333
5000		33.68953333	10.1178	3.424906667	1.78251
10000		134.708	40.32296667	13.175	7.0461
20000		532.1493333	159.8773333	52.9057	28.38436667

50 x 50 x 50 Volume – 4 Node Simulation – Also Only Intel Gold 6230

Future Improvements

- Dynamic particle color based off of velocity
- As seen in runtime results, the inability to use a more efficient, graph/grid backend for density calculation significantly hit performance
- Merging MPI Simulation and OpenGL rendering would remove significant, unnecessary communication. However, would require full control of development environment, like a Homelab, instead of through CCR to see video output
- Following said simulation and rendering merge, real-time adjustment of simulation constants (e.g. pressure and viscosity multipliers, particle count, particle size)

MPI Version 5 Specification, MPI Forum
https://www.mpi-forum.org/docs/mpi-5.0/mpi50-report.pdf

OpenMPI v5.0.0 Documentation, Open MPI Project
https://docs.open-mpi.org/en/v5.0.x/

Physics-Based Simulation and Animation of Fluids, Chand T. John PhD
https://unusualinsights.github.io/fluid_tutorial/

Fluid Simulation, Sebastian Lague
https://github.com/SebLague/Fluid-Sim/tree/Episode-01

OpenGL Wiki, OpenGL Wiki contributors
https://wikis.khronos.org/opengl/index.php?title=Main_Page&oldid=14430

References and Resources

Navier-Stokes Equations, Wikipedia
https://en.wikipedia.org/wiki/Navier%E2%80%93Stokes_equations

Computational Fluid Dynamics, Wikipedia
https://en.wikipedia.org/wiki/Computational_fluid_dynamics

References and Resources